REPORT DOCUMENTATION PAGE

AFRL-SR-BL-TR-01-

0423

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
		Final Technical Report 1 Dec 99 - 30 Nov 00

4. TITLE AND SUBTITLE
Dynamic Spectrum Allocation Algorithms

5. FUNDING NUMBERS
F49620-00-1-0011

6. AUTHOR(S)
Bala Kalyanasundaram

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
University of Pittsburgh
Office of Research
350 Thackeray Hall
Pittsburgh, PA 15260

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
AFOSR/NM
801 N. Randolph St, Rm 732
Arlington, VA 22203-1977

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
F49620-00-1-0011

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION AVAILABILITY STATEMENT
Approved for public release; distribution unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The setting for the spectrum allocation problems that we are currently considering consists of n tasks (i.e. flight tests). Each task has a bandwidth requirement, and a length. Thus each test can be thought of as a rectangle, with the vertical height of the rectangle being the bandwidth requirement. The scheduling space is a larger rectangle space, with height equal to the total spectrum available and length equal to the time period to be scheduled (e.g. a day). In order to avoid interference, two tasks must be placed in such a way that they do not overlap in this space. See figures 1 and 2 for example schedules. Additionally we assume that for each task there is a range of times at which it may be scheduled (e.g. it may be specified that a task should be scheduled between 10AM and 3PM), and an optional integer benefit/priority for the schedule (in our tests to date, all jobs have had equal priorities).

20010810 103

14. SUBJECT TERMS	15. NUMBER OF PAGES
	8
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

F49620-00-1-0011

Dynamic Spectrum Allocation Algorithms: Technical Report for AFOSR grant F49620010011

Bala Kalyanasundaram and Kirk Pruhs
University of Pittsburgh

Introduction

The setting for the spectrum allocation problems that we are currently considering consists of n tasks (i.e. flight tests). Each task has a bandwidth requirement, and a length. Thus each test can be thought of as a rectangle, with the vertical height of the rectangle being the bandwidth requirement. The scheduling space is a larger rectangle space, with height equal to the total spectrum available and length equal to the time period to be scheduled (e.g. a day). In order to avoid interference, two tasks must be placed in such a way that they do not overlap in this space. See figures 1 and 2 for example schedules. Additionally we assume that for each task there is a range of times at which it may be scheduled (e.g. it may be specified that a task should be scheduled between 10AM and 3PM), and an optional integer benefit/priority for the schedule (in our tests to date, all jobs have had equal priorities).

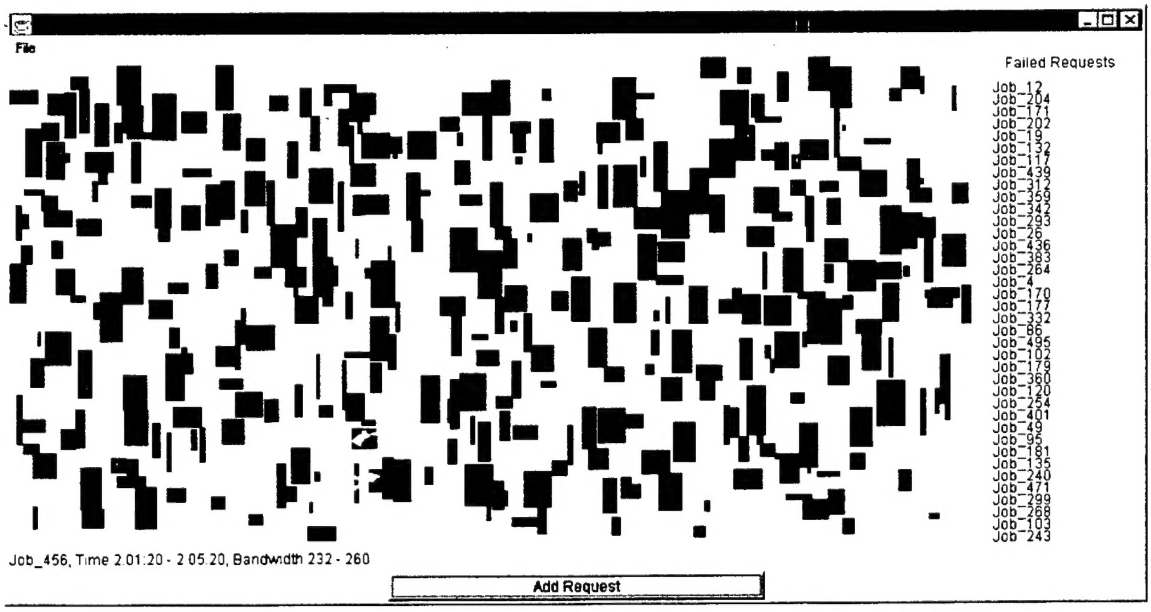

Figure 1: The schedule of 319 intervals scheduled by the algorithm Random simulating untrained human placement

Figure 2: The schedule of 459 intervals scheduled by the algorithm heuristic algorithm Timeline

We consider two settings. In the offline setting, the scheduler is given all the tasks at one time. For example, the scheduler is given all the requests for tests on a particular day a week away and must produce a schedule for that day. In the online setting, the scheduler is given tasks one by one, and must either schedule, or reject the tasks, as they arrive. The online setting models the case that some requests arrive a 2 weeks ahead of time, and some arrive 1 week ahead of time, etc. And the scheduler must give an answer to the requests as they arrive. In our tests to date, we have assumed that the scheduler's goal is to maximize the number of scheduled tasks.

Computing the optimal spectrum allocation is a known computationally infeasible problem. Thus for instances of even moderate size, one needs to fall back to computing near optimal schedules. We are currently conducting theoretical research on algorithms for spectrum allocation (e.g. mathematically analyzing the performance guarantees of various algorithms), and conducting experimental/empirical research on various algorithms in the literature, and on algorithms that we have designed.

The general goal of any good scheduler is to avoid fragmentation of the scheduling space. Roughly speaking, fragmentation is when the unused space is divided into lots of small regions (as opposed to a few larger regions). Fragmented space is less useful since future moderate sized requests can be fit into the small regions, and thus may be rejected.

One of our initial goals was to determine to what extent simple heuristics could improve upon untrained independent human schedulers. We modeled an

untrained human scheduler as follows. We assume that the tasks are scheduled in an arbitrary order, and that each task is placed randomly in the schedule. At first blush this may seem overly pessimistic, since one might think that humans would produce a placement better than random. But anecdotal evidence suggests that just the opposite might be true. That is, the placement that is easiest for a human to detect visually, is one in the center of a large unscheduled region, but this is also exactly the sort of strategy that will maximize fragmentation.

We now summarize our initial findings (we go into more detail in subsequent sections). The standard online scheduling method to avoid fragmentation is called First Fit, which roughly speaking tries to schedule every task as early in time and as low in the spectrum as possible. We find that for the input distributions that we tested, First Fit does not perform appreciably better than random placement. We then considered offline algorithms that consider the jobs in some predefined order. We also give some theoretical evidence of the difficulty of producing good schedules in the online setting[2]. More precisely, we showed that if one is given a sequence of tasks that can be fit into a spectrum of size B, there is no online algorithm that can fit these tasks into spectrum of size c*B, for any constant c. In the offline setting, we have found that, among the simple heuristic algorithms, the best are those that in some sense try to schedule the jobs from earliest in time to latest in time. On the inputs we tested these algorithms were consistently able to schedule 5% to 10% more jobs than Random. One can easily see difference visually in figures 1 and 2, which represent the schedules produced by Random and a heuristic that we call Timeline, respectively, on a randomly generated instance of 500 intervals. While a 5% to 10% improvement may seem modest, one needs to ask oneself about the benefit of being able to schedule a few more tests per day relative to the cost of the system required to produce such a schedule.

There are many scheduling problems that arise in manufacturing settings. In these settings, the method that seems to produce the best schedules is to have a trained human scheduler using a computer system as a tool. The system suggest an initial reasonable schedule, and the human can manipulate the schedule, probably also using the computer system to investigate possibilities (e.g. the human could query the system to see if you schedule more tasks if swapped the placement of two particular tasks.) We have developed a visual interface displaying the schedules (from which figures 1 and 2 were produced) using the Java programming language. It would be relatively easy to add an interface to allow the user to manipulate the schedule.

There are two obvious deficiencies to the simple heuristics that we have tested to date. First, jobs that that use more resources (e.g. the large bandwidth jobs, the longer jobs, and the lower laxity jobs) should be intuitively be scheduled first if they are going to be scheduled; That is, jobs that are harder to place, should be placed first. On the other hand, if not all jobs can be scheduled, the obvious jobs

to reject are those that use most resources. Thus these two heuristic observations conflict on whether high resources jobs should be considered early or late in the scheduling process. Second, there are several parameters that make a job desirable, e.g. short duration, low bandwidth requirement, high benefit, and these parameters may be conflicting. For example, it is not clear which is preferable, a short-duration high-bandwidth job, or a long-duration low-bandwidth job.

Recently some algorithms have been proposed in the literature that surmount these deficiencies [1, 3]. These algorithms have several additional benefits. One is that easily generalize to arbitrary integer benefits. Another is that allow for several scheduling ranges for each job, e.g. it is allowable to specify that a job should be scheduled Monday AM, Tuesday PM, or on Thursday. In these papers it was proven that the benefit achieved by the schedules produced by these algorithms is at least a constant fraction of the optimal benefit. Unfortunately the constants are too large to be considered good in practice. However, these algorithms have to date no been empirically/experimentally tested. We believe that these algorithms will likely perform much better in practice. We will test this hypothesis experimentally.

In the rest of the paper, we described the input distributions on which we tested the algorithms, the algorithms tested, and the results.

Input Distributions

The first input distribution that we one where essentially all parameters are uniformly generated from the range of possible values.

Input Distribution A:

Parameters:

B = total spectrum
T = total schedule time
N = Number of tasks
L = laxity. Laxity is the difference between the earliest time that a task can be scheduled and the latest time that a task can be scheduled.
X = maximum task length
B = maximum task bandwidth

Each task is generated in the following way: The bandwidth of the task is selected uniformly at random from the range [1, B]. The length of the task is selected uniformly at random from the range [1, X]. The laxity of the task is selected uniformly at random from the range from 0 to L times the length of the job. The earliest time that the task can be scheduled is selected uniformly at random from the range [0, T].

This input distribution generally assumes that all values of the parameters are equally likely to occur in practice. One difficulty with this distribution is that, while one can compare various candidate algorithms against each other, one generally can not determine how the schedules produced by the candidate algorithms compare against the optimal schedules. This is because the problem of computing the optimal schedules is a known infeasible problem (technically it is an NP-hard problem).

The standard method for circumventing this difficulty is to generate input instances with known optimal solutions. We adopt this approach for our second input distribution. Basically, the idea is to take a large rectangle, and cut it into smaller sub-rectangles that will be the tasks. The tasks' placements in the larger rectangle represent the optimal solution. To attempt to confuse the candidate algorithms, we added randomly generated time ranges when these tasks can be scheduled and feed them to the candidate algorithms in some random order.

INPUT DISTRIBTION B:

Parameters:

- B = total spectrum
- T = total schedule time
- N = number of tasks
- L = laxity parameter
- C = Bandwidth of original rectangle
- S = length of original rectangle

Each task is generated in the following way: We start with one task that takes up the entire time S and bandwidth C. We then recursively go through the list of current tasks. For each recursion, we take all of the tasks generated by the previous level and divide them roughly in half. The division alternates every round between a bandwidth division and a length division. The first division is always a bandwidth division. This allows the durations of the job set to be less regular (otherwise many jobs will either start or end along a single point in time). The actual division takes place uniformly in random in the middle half of the task to be divided (this guarantees that each partition has at least 1/4 of the original task). Also, if the input is too small to be divided (i.e. the side to be divided is less than or equal to 3 units), the job is not divided. At the end of every round, the job order is randomized. This recursive process will continue until N jobs are created.

The current placement is an optimal schedule. We have already set the length and bandwidth of each job. We now set the release date and deadline. The laxity l is selected uniformly at random from the range 0 to the length of the job times the laxity parameter L. A numbers p is generated uniformly at random from the range 0 to the laxity l of the job. The earliest time that the task can then be

scheduled is then p time units before the job's placement in the optimal schedule (or zero, which ever is larger).

The Greedy Algorithms

The algorithms simple heuristic algorithms that we tested are as follows:

Random: This is an online algorithm that considers the tasks in an arbitrary order. All of the possible places where the current task fits in the current schedule are found. The final assignment of the tasks is selected uniformly at from these placements. If no valid placement exists, the task is rejected. This algorithm is meant to model a non-careful human scheduler.

First Fit: This is the standard online heuristic to avoid fragmentation. The task is scheduled at the earliest time that is can be scheduled, and then at this time the task is scheduled as low as possible in the spectrum. Thus intuitively this algorithm tries to place every task as low and to the left as possible. If a job can not be scheduled, then it is rejected.

We also consider a collection of offline greedy algorithms. These greedy algorithms order the jobs in some particular order, and then process the jobs in this order. Each job is then scheduled, or rejected, using First Fit.
The sorting keys that we tried are:

- Earliest deadline first
- Largest bandwidth first
- Smallest bandwidth first
- Largest laxity first
- Smallest laxity first
- Largest laxity/length first
- Smallest laxity/length first

Timeline: This algorithm moves down the timeline of the request area. At every point of time on the timeline, the algorithm checks to see if an unscheduled task can be scheduled at this time. If more than one can be scheduled, the one with the earliest deadline is scheduled first. The selected job is then scheduled as low in the spectrum as possible.

Generally speaking the best algorithm among the ones that we tested was the Earliest Deadline First algorithm (although Timeline was essentially as good as Earliest Deadline First). We believe that the reason for this is that these algorithms essentially ignore the resource requirements of a job.

Experimental Results

We now give more detailed account of the results. The most important feature of these results is that Earliest Deadline First schedules 5% to 10% more intervals than does Random. Other unsurprising trends can also be seen. For example, larger the laxity parameter L, which specifies the flexibility of when tasks can be scheduled in time, the better the schedules produced. Another is that the lower the load parameter C in distribution B is, the more tasks are scheduled. A couple of cautionary notes are in order. First, note that the percentages in figure 3, it may not be the case one can optimally schedule 100% of the tasks in these tests. Second, note that the percentages are only really useful for comparison purposes. The actual percentages that will be seen in practice depends upon the input distributions.

	L = 1/2	L = 1	L = 2	L= 4
Random	67%	68.5%	75%	74.5%
First Fit	70%	72.5%	74%	75.5%
Earliest Deadline First	77%	80%	83%	83.5%
Best of Remaining Algorithms	74%	76.5%	80%	83.5%

Figure 3: Results for input distribution A with N=200

	L = 1/2	L = 1	L = 2	L= 4
Random	69.6%	72%	74.4%	70.8%
First Fit	67.6%	72.4%	75.2%	77.6%
Earliest Deadline First	75.6%	81.2%	84.5%	86%
Best of Remaining Algorithms	74%	79.6%	84.8%	88%

Figure 3: Results for input distribution B with N=250

	L = 1/2	L = 1	L = 2	L= 4
Random	78%	75.2%	81.6%	76.5%
First Fit	76.8%	77.2%	79.6%	80%
Earliest Deadline First	83.6%	85.6%	87.6%	89.6%
Best of Remaining Algorithms	82.4%	85.6%	86.8%	90%

Figure 3: Results for input distribution C with N=250

References:

1. A. Barnoy, R. Bar-Yehuda, A. Freund, J. Naor, and B. Schieber, "A unified approach to approximating resource allocation and scheduling", ACM Symposium on Theory of Computing, 2000.

2. B. Kalyanasundaram, and K. Pruhs, "Dynamic spectrum allocation: the impotency of duration notification", special issue of *Journal of Scheduling* devoted to approximation algorithms, **3**(5), 289 - 296, 2000

3. S. Leonardi, A. Marchetti-Spaccamela and A. Vitaletti, "Approximation algorithms for bandwidth and storage allocation problems under real time constraints", Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS), 2000.